XML Basics

This module describes XML (eXtensible Markup Language) and the rules that govern its usage. It also explains what a well-formed and valid document is.

What is XML?

The eXtensible Markup Language (**XML**) is a **meta-markup language** defined by the World Wide Web Consortium (W3C). It is not strictly a markup language itself, but rather a set of rules for creating markup languages. For our purposes a **markup language** is any language (HTML, for example) that uses tags surrounding text to convey information such as content or format. CNXML, the markup language used by the Connexions Project is an example of a language written in XML. There are many other examples at the W3C site. Here is an example of some markup in CNXML.

Example:

```
<para>
  This is a paragraph in <term>CNXML</term>.
Notice that the markup
  contains tags that express the meaning of the
text.
</para>
```

`<para>` and `</para>` are the tags that enclose the text. In XML, tags are always marked by angle brackets (also known as `<` and `>`). **Tags** generally come in pairs. An opening tag will look like `<tagname>`. A closing tag will look like `</tagname>`, with a `/` preceding the tag name.

XML allows the separation of presentation from content. For example, HTML has tags such as `<u>` and `<i>`, which underline and italicize text respectively. This does not express content information, only formatting. XML allows you to define your own language of tags to represent content. You could create a tag called `<book>` to represent book titles, and create a stylesheet (a separate formatting document), that says that every `<book>` tag should be italicized or underlined. Then when you want to change the presentation of that type of content, you just change one small part of the stylesheet. Also, if you make tags that convey the content of the document, you can enable better searching. For example, you might look for the author of a document by looking at the author tag.

Well-formed XML

XML has a few rules that apply to all of its languages, including CNXML. If a document satisfies these rules, then it is **well-formed**. XML documents are required to be well-formed.

- Every tag that is opened must be closed. An opening tag looks like `<module>` and a closing tag looks like `</module>`. There is a shortcut. If your tag contains no other tags (referred to as an **empty tag**), then you can can type a / before the end of the opening tag and delete the closing tag. For example, `<media> </media>` can be abbreviated `<media/>`.
- Tags must be nested within each other. So, `<b>red <i>and</i> blue</b>` is fine, but `<b>red <i>and</b> blue</i>`is incorrect because the `<b>` and `<i>` tags have overlapping content.
- You must put either single or double quotes around an attribute value. An **attribute** is some sort of information that is associated with a tag and is listed inside of the tag itself. For example, `<module id="m0001">` and `<module id='m0001'>` are fine, but `<module id=m0001>` is incorrect.
- You can also choose to start every document with an **XML declaration**. If you do use the XML declaration, then it has to be the very first thing in the file. It cannot even be preceded by whitespace. It is not considered to be a tag. The XML declaration is as follows. `<?`

`xml version="1.0"?>` You can also include other information such as the encoding of the document or whether the document depends on other files or not.
- There must be one tag that contains all of the other tags. For example in xhtml `<html>` and `</html>` must surround all of the other tags. There are some things that are included at the top of the document that are not tags and that are not included with the tags. The XML declaration is an example of this.

Valid XML

It is possible to define a set of rules that apply to all of the tags in a particular XML language. These rules can be defined in a couple of different ways. The most common way is to use a **DTD** (Document Type Definition). Any document which follows all of the rules for that language is called **valid**. A document is not required to be valid in order to be XML. However, it is generally a good idea.

Entity References

Note:
Entity References in CNXML 0.6
Entity references are no longer supported by CNXML 0.6. Instead, we suggest that you use character references as described below to add special characters to your module.

XML uses several characters in special ways as part of its markup, in particular the less-than symbol (<), the greater-than symbol (>), the double quotation mark ("), the apostrophe ('), and the ampersand (**&**). You've already seen examples of markup using the first four of those previously in this module. But what if you need to these characters in your content, and you don't want them to be treated as part of the markup by XML processors? You can use XML **entity references** for this purpose. The

XML Specification defines the following five entity references for use in any well-formed XML document:

- `&amp;` refers to an ampersand (&)
- `&lt;` refers to a less-than symbol (<)
- `&gt;` refers to a greater-than symbol (>)
- `&quot;` refers to a double-quote mark (")
- `&apos;` refers to an apostrophe (')

Example:
Suppose you have a document with the following: `<para id="p1">The firm was known as Scrooge and Marley.</para>` you could replace 'and' with the entity reference `&amp;`: `<para id="p1">The firm was known as Scrooge &amp; Marley.</para>`

All entity references outside the above five must be defined in a document type declaration, and they may only be used in documents that conform to that DTD. Note that an entity reference always begins with `&` and ends with `;`.

Character References

You can also use any character defined in **Unicode** in an XML document by means of **character references**. Unicode is a project to define a unique code for every character in any human language. Unicode is very useful any time that you need to use a symbol that is not a part of ASCII.

Character references in XML either begin with `&#`, or they begin with `&#x`, and they end with a semicolon `;`. A character reference contains a representation of a Unicode code point: if it begins with `&#`, then it contains a decimal representation of a Unicode code point; if it begins with `&#x`, then it contains a hexidecimal representation of a Unicode code point.

Example:
The hexidecimal representation of the Unicode code point for the small 'o' with a stroke is `00F8`, and the decimal representation for the same is `248`. Therefore, the character references for the small 'o' with a stroke are `&#x00F8;` and `&#248;` So you could write `<emphasis>The majestik m&#x00F8;&#x00F8;se</emphasis>` or `<emphasis>The majestik m&#248;&#248;se</emphasis>` or even `<emphasis>The majestik m&#x00F8;&#248;se</emphasis>` to get " The majestik møøse "

Combining XML Languages
This module explains how to use XML namespaces and DTDs to combine multiple XML languages in the same document.

XML allows you to create documents in custom markup languages. But what if you want to combine markup from multiple languages in the same document? What if there are one or more tags that exist in both languages, but with different meanings? You could, for example, have a `<table>` tag in HTML and one in a language describing office furniture as well. How do you use these tags unambiguously, without losing functionality?

The solution is to use an extension to XML called **namespaces** (See the W3C's recommendation, Namespaces in XML). A namespace associates a unique global identifier (usually a URI) with a particular set of tags and their usage rules. To declare a namespace for a particular tag, set the `xmlns` attribute to the value of the unique identifier.

You can also define a **namespace prefix** for use in your document. To do this, use a modified version of the `xmlns` attribute. For example, you would use the attribute `xmlns:foo="http://somewhere.org/foo"` to associate the prefix `foo` with the namespace identifier `http://somewhere.org/foo`. You can then indicate which tags come from that namespace by adding the appropriate prefix to each tag. Thus, the `bar` tag in foo's namespace would be written as `<foo:bar>` and `</foo:bar>`.

When you use the default namespace any children of that tag lacking an explicit prefix will be assumed to have come from the same namespace. This allows you to define a default namespace for all of the children of a tag. This is especially useful when used on the **root node**, which is the outermost tag in a document.

Example:
For CNXML 0.6 there is only one schema. The document tag will contain the namespace for all available languages and will look like this:

```
    <document xmlns="http://cnx.rice.edu/cnxml"

xmlns:md="http://cnx.rice.edu/mdml/0.4"
              xmlns:bib="http://bibtexml.sf.net/"

xmlns:m="http://www.w3.org/1998/Math/MathML"

xmlns:q="http://cnx.rice.edu/qml/1.0"
              id="new"
              cnxml-version="0.6"
              module-id="new">
```

The Basic CNXML
This is a basic introduction to the CNXML language. It includes a description on how to begin a CNXML module and also examples of the basic tags needed to start writing in CNXML.

Starting with CNXML

CNXML is a lightweight XML markup language for marking up educational content. The goal of CNXML is to convey the content of the material and not a particular presentation. Connexions uses the Connexions Markup Language (CNXML) as its primary language for storing documents. Now let's get started!

CNXML Tags

Document

All CNXML documents have as their root the `document` tag. Everything about the document including it's metadata and content are contained within the document tag. It is important that you understand the basic structure for a CNXML document. The structure is as follows:
Document (root tag)

- **Title**
- **Metadata Section**
- **Content Section**

The document tag has one required attribute:

- `id` - a unique ID given to the document.

This is automatically assigned.

ID Requirements

One major difference between CNXML and other markup languages is the `id` attribute requirement. Certain tags require that you include the `id` attribute, but all can possess an `id`. The tags requiring an id are listed below:

- document
- para
- equation
- list
- rule
- definition
- exercise
- table
- div
- section
- subfigure
- example
- footnote
- problem
- solution
- block quotes
- media
- meaning
- proof
- list
- preformat
- block code
- figure
- block notes

So, if you are going to use any of the above tags, be sure to add the `id` attribute and give it a unique 'id'. Be aware that in CNXML 0.6 ids will be generated automatically, but you are still permitted to specify you own ids if you wish.

Example:
Here is an example of a couple of paras containing a user generated ids.

```
<para id='uniqueid1'>
  This is an example to illustrate the use of the
<code>id</code> attribute.
</para>
<para id='uniqueid2'>
  This paragraph has a different id than the last.
</para>
```

Note: Any tag can contain an `id` attribute. This is useful if you want to link to the information contained in a particular tag.

Namespaces

The `document` tag should also contain any **namespace** declarations. Namespaces allow us to easily use other mark-up languages within CNXML without having to worry about whether tag name collision will occur. For simple documents using only CNXML, you need to include the CNXML namespace attribute . Any additional languages need to be declared as well and should be given their own prefixes. For example, to associate the MathML namespace with the prefix "m", include the following attribute: `xmlns:m='http://www.w3.org/1998/Math/MathML'`. This states that any tag with a prepended "m" will be interpreted as a MathML tag while any tag without a prefix will be interpreted as CNXML. The document tag should also contain the metadata namespace `xmlns:md="http://cnx.rice.edu/mdml/0.4"`, the bibtex namspace `xmlns:bib="http://bibtexml.sf.net/"`, and the question markup language `xmlns:q="http://cnx.rice.edu/qml/1.0"`.

Example:
This what the document tag should look like.

```
<document xmlns="http://cnx.rice.edu/cnxml"
xmlns:m="http://www.w3.org/1998/Math/MathML"
xmlns:md="http://cnx.rice.edu/mdml/0.4"
xmlns:bib="http://bibtexml.sf.net/"
xmlns:q="http://cnx.rice.edu/qml/1.0" id="m9000"
module-id="" cnxml-version="0.6">
```

Note: Be aware that the you document id can not be the same as this example. Each module will have its own unique id.

Title

The `title` tag can be used with many CNXML tags to hold the name of its parent. This tag can only contain information in ASCII text or MathML. I mention it here to allow you to put in the name of the module (since I mentioned that it was the first required tag in the document tag).

Example:

```
<title>Grilling a Good Steak</title>
```

Note: Please see the CNXML tag list in Edit-In-Place to see if a tag can be named.

Content

Now that you have the `document` tag set up with an id and namespace info, the next thing to do with your document is add content. By 'content' I mean the text that will make up the bulk of your document.

Note: Strictly speaking the metadata should precede information about content, but we will leave this until later.

All of this content is conveniently placed in the `content` tag. Every CNXML document will have one `content` tag. The body of the document will be here inside the `content` tag.

Structural tags are the tags which are used inside of the `content` tag to give structure to the document. These tags are discussed below.

Structure Tags

Some of the structure tags are section, para, document, and content.

We have already discussed the document and content tags, so we will proceed with a short description and examples of the other other tags listed.

Para

Text can be inserted into documents by using the `para` tag. Each para has a required `id` which must be unique within the document.

Example:

```
<para id='intro'>
  I have eaten many steaks in my life and none 
have been more satisfying
  than the backyard-grill cooked steak.  Maybe 
this is because of the
  relaxing nature of drinking a beer, being 
outside, and lounging that
  accompanies the grilling procedure.  Maybe it is 
because of the aroma
  of the grill and the beef perfectly seasoned to 
your taste.  Either
  way, this document shows how a good steak can be 
prepared.
</para>
```

Section

As often is the case in textbooks, chapters are divided into smaller sections. Because it is often necessary to segment text for better understanding and coherence, CNXML has included a `section` tag.

The section tag has one required attribute, `id`, and a optional first child tag, title.

Example:

```
<section id='ingredsec'>
  <title>Ingredients</title>
    <list> ... </list>
</section>
<section id='marinadesec'>
  <title>Marinade</title>
  <para id='marinate'> ... </para>
    <list id='marinade'> ... </list>
  <para id='tobecontinued'> ... </para>
</section>
<section id='grillingsec'>
  <title>Grilling</title>
  <para id='prepgrill'> ... </para>
  <para id='grilling'> ... </para>
</section>
```

Obviously ellipses would be replaced by appropriate text.

Inline Tags

Inline tags are used to embed content and functionality inside of the structural tags. Some of the more commonly used tags are discussed below.

Emphasis

The `emphasis` tag is used to emphasize text in a CNXML document where emphasis in text would be needed or desired. It is important to note that this refers to **semantic** emphasis and not a typeface, although many stylesheets may choose to render it visually with a different typeface.

Example:

```
<para id='intro'>
  I have eaten many steaks in my life and none
have been more satisfying
  than the backyard-grill cooked steak.  Maybe
this is because of the
  relaxing nature of drinking a beer, being
outside, and lounging that
  accompanies the grilling procedure.  Maybe it is
because of the aroma
  of the grill and the beef
<emphasis>perfectly</emphasis> seasoned to
  your taste.  Either way, this document shows how
a good steak can be
  prepared.
</para>
```

Term

The `term` tag is used to mark words or phrases which are being defined. However, its use is confined to either a para or definition tag. The `term` tag has several optional attributes:

- `url` - a URL specifying the source or definition of the term.
- `window` - contains the possible values "replace" which results in the associated url opening in the present window, and "new" which result in the associated url opening in a new window or tab.
- `document` - the id of another Connexions module or collection.
- `target-id` - the id of a specific element (such as a para or section) in the current or another Connexions document.

- `resource` - This reference points to a file that is associated with the term in question. The resource could be a pdf, text file, or any other supplementary resource.
- `version` - The version of a Connexions module or collection. This attribute is used in conjunction with the document attribute.
- `id` - A unique identifier, whose value must begin, with a letter and contain only letters, numbers, hyphens, underscores, colons, and/or periods (no spaces).

Example:

```
<para id='marinade'>
  To ensure the best flavor possible, it is
necessary to marinate the
  beef.  A steak <term
url='http://marinade.com'>marinates</term> when
  left to sit in a prepared sauce, or
<term>marinade</term>, where it
  will absorb the flavors of the ingredients.
Marinating may take as
  little as 15 minutes or as long as 6 hours and
should
  <emphasis>always</emphasis> be done in the
refrigerator and
  <emphasis>not</emphasis> at room temperature.
</para>
```

Note

The `note` tag creates a note to the reader, which could be a warning, tip, etc. There are five allowed types of note: note aside warning tip important .

The type of note is specified by an optional `type` attribute.

Example:

```
<para id='intro'>
  I have eaten many steaks in my life and none
have been more
  satisfying than the backyard-grill cooked steak.
Maybe this is
  because of the relaxing nature of drinking a
beer, being outside,
  and lounging that accompanies the grilling
procedure.  <note
  type='warning'>Excessive drinking or fun may
result in overcooked or
  burned steak.</note> Maybe it is because of the
aroma of the grill
  and the beef <emphasis>perfectly</emphasis>
seasoned to your taste.
  Either way, this document shows how a good steak
can be prepared.
</para>
```

The above markup will display as:
I have eaten many steaks in my life and none have been more satisfying than the backyard-grill cooked steak. Maybe this is because of the relaxing nature of drinking a beer, being outside, and lounging that accompanies the grilling procedure.

Note: Excessive drinking or fun may result in overcooked or burned steak.

Maybe it is because of the aroma of the grill and the beef **perfectly** seasoned to your taste. Either way, this document shows how a good steak can be prepared.

Link

The `link` tag is used to provide a quick link to other Connexions modules, collections or external websites. The link tag can contain the following attributes.

- `strength` - The Strength attribute can contain the values 1, 2, or 3 (with 3 being the strongest) specifying the relevance of the link.
- `url` - a URL specifying the source or definition of the term.
- `window` - contains the possible values "replace" which results in the associated url opening in the present window, and "new" which result in the associated url opening in a new window or tab.
- `document` - the id of another Connexions module or collection.
- `target-id` - the id of a specific element (such as a para or section) in the current or another Connexions document.
- `resource` - This reference points to a file that is associated with the term in question. The resource could be a pdf, text file, or any other supplementary resource.
- `version` - The version of a Connexions module or collection. This attribute is used in conjunction with the document attribute.
- `id` - A unique identifier, whose value must begin, with a letter and contain only letters, numbers, hyphens, underscores, colons, and/or periods (no spaces).

The `target` and `document` attributes can be used together or alone. If both are used then you will link to a particular tag in another document. If only `document` is used, you will link to another document. If only `target` is used, you will link to a particular tag within the current document.

Cite

The `cite` tag is used to refer to non-electronic materials within a document, primarily containing the title of a work. Cite has several optional attributes:

- `url` - a URL specifying the source or definition of the term.
- `window` - contains the possible values "replace" which results in the associated url opening in the present window, and "new" which result in the associated url opening in a new window or tab.
- `document` - the id of another Connexions module or collection.
- `target-id` - the id of a specific element (such as a para or section) in the current or another Connexions document.
- `resource` - This reference points to a file that is associated with the term in question. The resource could be a pdf, text file, or any other supplementary resource.
- `version` - The version of a Connexions module or collection. This attribute is used in conjunction with the document attribute.
- `id` - A unique identifier, whose value must begin, with a letter and contain only letters, numbers, hyphens, underscores, colons, and/or periods (no spaces).

Quote

The `quote` tag is used to denote that some text is a direct quote from some other source. The quote tag has a `display` attribute which denotes whether the quote is `inline` or `block`. `Quote` can also contain all of the attributes associated with cite.

Example:
```
<para id='steakquote'> Everyone has an opinion on
how a steak should be cooked. <quote
display='inline'>"A good steak should be pink in
the middle and black on the outside."</quote>
```

`Although this may sound reasonable many remember the words of George Washington: <quote type='block'>"In any free country a man should have the ability to purchase a nice rare steak." </quote> </para>` Everyone has an opinion on how a steak should be cooked. ""A good steak should be pink in the middle and black on the outside."" Although this may sound reasonable many remember the words of George Washington: ""In any free country a man should have the ability to purchase a nice rare steak.""

Foreign

The `foreign` tag is used to denote that a foreign word or phrase is being used. `Foreign` can also contain all of the attributes associated with cite.

Example:
`<para id='steakquote2'> In many latin american countries steak is called <foreign>carne asada</foreign>. </para>` In many latin american countries steak is called carne asada.

Document Example Code

Below is an example of what your document could look like if you included all the tags above to make a document about making a steak.

```
<document id='meat'>
<title>Grilling a Good Steak</title>
```

```
      <content>

        <section id='intro'>
          <para id='intro'>
            I have eaten many steaks in my life
and none have been more
            satisfying than the backyard-grill
cooked steak.  Maybe this is
            because of the relaxing nature of
drinking a beer, being
            outside, and lounging that accompanies
the grilling procedure.
            <note type='warning'>Excessive
drinking or fun may result in
            overcooked or burned steak.</note>
Maybe it is because of the
            aroma of the grill and the beef
<emphasis>perfectly</emphasis>
            seasoned to your taste.  Either way,
this document shows how a
            good steak can be prepared.
          </para>
        </section>

        <section id='marinate_section'>
          <para id='marinate'>
            To ensure the best flavor possible, it
is necessary to marinate
            the beef.  A steak
<term>marinates</term> when left to sit in
            <term>marinade</term>, or prepared
sauce, where it will absorb
            the flavor of the ingredients.
Marinating may take as little as
            15 minutes or as long as 6 hours and
should
            <emphasis>always</emphasis> be done in
```

```
the refrigerator and
            <emphasis>not</emphasis> at room
temperature.
          </para>
        </section>

        <section  id='tobecontinued_section'>
          <para id='tobecontinued'>
            I'll be adding to this document in
<link document='m9006'
            >The Intermediate CNXML</link> which
focuses on more
            advanced CNXML tags.  For more
marinades see the <link
            url='http://www.2eatcab.com'>Angus
Beef website</link>.
            Finally, a good resource is the
<cite>Steak Lover's Cookbook --
            William Rice</cite>.
          </para>
        </section>

      </content>
      </document>
```

See how Connexions would render this example.

Other Required Stuff

The first line in any XML file should be the XML declaration. (Strictly speaking, this is optional, but it's a good practice to follow). The XML declaration looks like this: `<?xml version="1.0" encoding="utf-8"?>`, and must not be preceeded by any blank lines or whitespace. CNXML 0.6 only uses one schema, so there is no need to specificy specific DTDs. Below is an example of a correct CNXML 0.6 document tag containing the proper namespaces.

Example:

```
<document xmlns="http://cnx.rice.edu/cnxml"
xmlns:m="http://www.w3.org/1998/Math/MathML"
xmlns:md="http://cnx.rice.edu/mdml/0.4"
xmlns:bib="http://bibtexml.sf.net/"
xmlns:q="http://cnx.rice.edu/qml/1.0"
id="m9000" module-id="" cnxml-version="0.6">
```

Conclusions

Remember that when composing documents it is always best to consult the CNXML Tag List for any questions regarding the exact usage of CNXML tags. For more advanced topics see The Intermediate CNXML or The Advanced CNXML, which concludes the cooking lesson.

The Intermediate CNXML
This is the second installment of the CNXML language tutorials. It is designed to give a more comprehensive look at the CNXML tags and explain some more advanced uses of the language. Includes helpful examples of CNXML tags, extending the module created in The Basic CNXML tutorial.

Example

As is often the case in textbooks, authors will include examples in the middle of a chapter or section. For this reason CNXML provides a tag that allows an author to include examples in a document. The example tag has a unique `id` attribute and can contain most tags as children, the first being an optional title. For specifics you should always consult the CNXML Spec.

Example:

```
<example id='tboneexam'>
  <figure id='tbonefig'>
    <title>T-Bone Steak</title>
     <media id="image-example" display="block"
alt="A T-bone Steak.">
       <image type='image/jpg' src='tbone.jpg'/>
     </media>
  </figure>
</example>
```

Figure

The `figure` tag provides the structure for creating a figure within a document. They can contain either two or more subfigure tags, or a single media, table, or code tag.

The `figure` tag has two attributes:

- `id` - a unique ID, required
- `orient` - defines how multiple subfigures are to be displayed. It takes two values, `vertical` or `horizontal`, and will default to `horizontal`.

The optional first tag of the `figure` tag is title which is used to title a figure.

The `title` tag is followed by any of the tags listed above; however, the most commonly used tag is media, which is used to include any sort of media such as images, video, music, or java applets. The media object tags have two required attributes:

- `src` - the location of the displayed media
- `mime-type` - defines the type of media being displayed, which can be any valid MIME type.
    - audio - audio/mp3, audio/wav, etc.
    - video - video/qt, video/mov, etc.
    - image - image/png, image/gif, etc.
    - application - application/PostScript, application/x-java-applet, etc.

The final tag is the optional `caption` which is used to add a small caption to the figure.

Example:

```
<figure id='tbone'>
  <title>T-Bone Steak</title>
  <media id="image-example" display="block" alt="A
T-bone Steak.">
     <image mime-type='image/jpeg'
src='tbone.jpg'/>
  </media>
  <caption>
    Upon successful completion of these documents,
you should be able
    to grill a steak that looks just as good!
  </caption>
</figure>
```

Subfigure

The `subfigure` tag is used when you want to include more than one media, code or table within the same figure.

The usage of the `subfigure` tag is similar to that of figure. It has an optional `id` attribute, an optional first child title tag, a single media, code or table, followed by an optional caption.

Now the `orient` attribute for figure becomes very important. `orient` lets you specify whether the subfigures should be displayed side-to-side or one on top of the other.

Example:

```
<figure orient='horizontal' id='horfig'>
<title>Steaks</title> <subfigure
id='subfigtbone1'> <title>T-Bone</title> <media
id="image-example" display="block" alt="A T-bone
Steak."> <image type='image/jpeg'
```

```
src='tbone.jpg'/> </media> </subfigure> <subfigure
id='subfingnystrip1'> <title>New York
Strip</title> <media id="image-example"
display="block" alt="A NY Strip."> <image mime-
type='image/jpeg' src='ny_strip.gif'/> </media>
</subfigure> <caption> Upon successful completion
of these documents, you should be able to grill a
steak that looks just as good! </caption>
</figure> Or <figure orient='vertical' id='verfig'>
<title>Steaks</title> <subfigure
id='subfigtbone2'> <title>T-Bone</title> <media
id="image-example" display="block" alt="A T-bone
Steak."> <image mime-type='image/jpeg'
src='tbone.jpg'/> </media> </subfigure> <subfigure
id='subfig2'> <title>New York Strip</title> <media
id="image-example" display="block" alt="A NY
Strip."> <image mime-type='image/jpeg'
src='ny_strip.jpg'/> </media> </subfigure>
<caption> Upon successful completion of these
documents, you should be able to grill a steak
that looks just as good! </caption> </figure>
```

List

The `list` tag is used to make lists. It has two attributes:

- `id` - a unique ID, required
- `list-type` - defines the formatting of the list. `list-type` takes the values `bulleted` (default), `enumerated`, `named-item` or `inline`

The `list` tag has two children: title, which is optional, and `item`, which is where the list information is stored.

Example:
Example List
`<list id='marinade' list-type='enumerated'> <title>Beer Marinade</title> <item>pour beer into large bowl</item> <item>add chili powder to taste</item> <item>squeeze half lime into beer marinade</item> <item>place steak in beer, let soak for 30 minutes</item> </list>` The resulting list will look like:
Beer Marinade

1. pour beer into large bowl
2. add chili powder to taste
3. squeeze half lime into beer marinade
4. place steak in beer, let soak for 30 minutes

Example:
New List Types Example
CNXML 0.6 gives you much more control over the list environment. Now you will be able to choose from eight preset bullet styles as well as an option that allows you to choose your own literal text to serve as the bullet style. The enumerated list type now offers several styles, including Arabic numerals, upper and lower case alphabet characters, and also upper and lower case Roman numerals. In addition to these changes, you can now also select to have your lists follow a stepwise progression. In CNXML 0.6 the named-item list has been slightly altered, and is now called a labeled-item list. As you may have guessed, the change is quite intuitive. Instead of using <name> to specify the item's label, you use <label>.
Here is an example of a stepwise enumerate list: `<list id="eip-165" list-type="enumerated" number-style="arabic" class="stepwise"> <title>Beer Marinade</title> <item>pour beer into large bowl</item> <item>add chili powder to taste</item> <item>squeeze half lime into beer marinade</item> <item>place steak`

```
in beer, let soak for 30 minutes</item> </list>
```
The resulting list will look like:
Beer Marinade

- **Step 1**pour beer into large bowl
- **Step 2**add chili powder to taste
- **Step 3** squeeze half lime into beer marinade
- **Step 4** place steak in beer, let soak for 30 minutes

Equation

The `equation` tag is used to set off and number equations in CNXML documents by using ASCII text, MathML and embedded media to display math.

Note: It is strongly encouraged, however, to use equation with MathML tags when displaying math.

ASCII Text and Images

The first child of equation is an optional title followed by any number of media tags.

Example:
```
<equation id="eqn14"> <title>Euler's
Relation</title> <media id="equation-example"
display="block" alt="Euler's Relation."> <image
mime-type='image/gif' src='euler.gif' /> </media>
</equation> <equation id='eqn15'> <title>Simple
```

`Arithmetic</title> 11+27=38 </equation>` This equation will display as:

Equation:

Simple Arithmetic

11+27=38

You could also write this equation using MathML: `<equation id="eqn22"> <m:math> <m:mn>11</m:mn> <m:mo>+</m:mo> <m:mn>27</m:mn> <m:mo>=</m:mo> <m:mn>38</m:mn> </m:math> </equation>`

Definition

The `definition` tag is used to define a word in a CNXML document. It has a required `id` attribute and three children: term, `meaning` and example. How to use definition is a little confusing, so don't forget to check out [link].

The first child tag should be term which contains the word/phrase being defined. It is then followed by a `meaning` tag which is followed by any number of examples. This process repeats for all meanings.

Example:

```
<definition id='tbonedef'>
  <term>T-Bone</term>
  <meaning>
    "The T-bone steak is cut between 1 and 3
inches thick and comes
     from the center section of the short loin.
This steak is
     characterized by its T-shape bone, has a
fine-grained shell and a
```

```
     small tenderloin eye,"

<cite>http://www.chophousecalgary.com/steak.html</
cite>.
  </meaning>
  <example id='tboneexam'>
    <figure id='tbonefig'>
      <title>T-Bone Steak</title>
        <media id="image-example" display="block"
alt="A T-bone Steak.">
          <image mime-type='image/jpeg'
src='tbone.jpg'/>
        </media>
      </media>
    </figure>
  </example>
</definition>
```

T-Bone

"The T-bone steak is cut between 1 and 3 inches thick and comes from the center section of the short loin. This steak is characterized by its T-shape bone, has a fine-grained shell and a small tenderloin eye," *http://www.chophousecalgary.com/steak.html*.

Example:
T-Bone Steak

Rule

The `rule` tag is used to insert a rule, such as a theorem, axiom, or rule of thumb, into a cnxml document. It has two attributes:

- `id` - required, unique ID
- `type` - required, specificies the type of rule (e.g. theorem, axiom, rule of thumb, etc.)

It may also have an optional title and it must have one or more statement tags and zero or more proof or example tags.

Statement

The `statement` tag is used inside a `rule` tag and defines the statement of the rule. It has an optional `id` attribute, which, like all IDs, must be unique. It also has two children, para and equation.

Proof

The `proof` tag is used inside the `rule` tag and marks the proof of the rule. It has an optional `id` attribute and may contain another rule, para, equation, figure, or list tag.

Using rule

Example:

```
<rule id='murph' type='law'>
  <title>Murphy's Law</title>
  <statement>
    <para id='murphp1'>
      If there are two or more ways to do
something, and one of those
      ways can result in a catastrophe, then
someone will do it.
    </para>
  </statement>
  <proof>
    <para id='murphp2'>
      Edward A. Murphy, Jr. was one of the
engineers on the
      rocket-sled experiments that were done by
the U.S. Air Force in
      1949 to test human acceleration tolerances
(USAF project
      MX981). One experiment involved a set of 16
accelerometers
      mounted to different parts of the subject's
body. There were two
      ways each sensor could be glued to its
```

```
mount, and somebody
      methodically installed all 16 the wrong way
around. Murphy then
      made the original form of his pronouncement,
which the test
      subject (Major John Paul Stapp) quoted at a
news conference a
      few days later

<cite>http://www.lylemariam.com/murphy.htm</cite>.
    </para>
  </proof>
</rule>
```

Murphy's Law

If there are two or more ways to do something, and one of those ways can result in a catastrophe, then someone will do it.

Edward A. Murphy, Jr. was one of the engineers on the rocket-sled experiments that were done by the U.S. Air Force in 1949 to test human acceleration tolerances (USAF project MX981). One experiment involved a set of 16 accelerometers mounted to different parts of the subject's body. There were two ways each sensor could be glued to its mount, and somebody methodically installed all 16 the wrong way around. Murphy then made the original form of his pronouncement, which the test subject (Major John Paul Stapp) quoted at a news conference a few days later *http://www.lylemariam.com/murphy.htm*.

Finishing Remarks

Thanks for making it through another tutorial. I'm sure that you still want to know more so here's a link to The Advanced CNXML.

The Advanced CNXML
This is the final installment of my three part tutorial on the CNXML language. It is currently valid for the most recent release of the language, CNXML 0.6. The keywords contain a list of the tags described in this tutorial. Along with the example code in this module there is also an example module that has been growing throughout the tutorial.

Code

The `code` tag is used to insert example computer output/input as either inline text within a paragraph or as a block of text. To see which tags it may contain or be inside, consult the CNXML Spec. The `code` tag has a `display` attribute with two possible values.

- `inline` (default) - used to specify code that is inline.
- `block` - used to specify code that should be in a separate block of text.

Example:

```
<para id='copy'>
  In a unix terminal the command to copy a file is <code
  display='inline'>cp original copy</code>.
</para>
```

In a unix terminal the command to copy a file is `cp original copy`

Exercise

The `exercise` tag provides a tag for authors to add practice problems into their documents. The `exercise` tag has a required `id` attribute and has two child tags, `problem` and `solution`.

To create more complex answers, such as multiple-choice, multiple-response, ordered-response, and text-response questions, QML (Questions Markup Language) may used in place of the problem and solution tags. For more information, please see the information about QML.

Example:

```
<exercise id='grilltest'>
  <problem>
    <para id='grilltestp1'>
      For food safety, a steak should be cooked to
a minimum
      temperature of what?
    </para>
  </problem>
  <solution>
    <para id='sol1p1'>
      160&deg; F or until the juices run clear and
the meat is no
      longer pink.
    </para>
  </solution>
</exercise>
```

Exercise:

Problem:

For food safety, a steak should be cooked to a minimum temperature of what?

Solution:

160° F or until the juices run clear and the meat is no longer pink

CALS Table

CNXML uses the industry standard CALS Table Model for including tables into CNXML documents. Provided below is a brief description of the CALS tags, their attributes, and children (along with a helpful example). For a more complete description of the CALS Table consult the CALS Table Spec.

table

The `table` tag marks the beginning of a table. It has an optional first child of title and must contain one or more tgroup tags. The `table` tag also has many attributes, to find out more information consult the CALS Table Spec.

tgroup

The `tgroup` tag marks the beginning of a new portion of a table. It has a required attribute `cols` which is the number of columns in the `tgroup`. Its children tags are zero, one, or more colspec or spanspec, zero or one thead or tfoot, and one tbody tag.

colspec

The `colspec` tag is an **empty tag** that specificies the column of a table or entrytbl. The names and numbers specified as attributes are used for referencing by other tags.

spanspec

The `spanspec` tag is an empty tag that identifies a horizontal span of columns and associated attributes that can subsequently be referenced by its spanname for repeated use in entry or entrytbl in different rows.

thead

The `thead` tag identifies the heading row of a tgroup or entrytbl. The `thead` tag can have zero, one, or more colspec tags and one or more row.

tfoot

The `tfoot` tag identifies the rows of footer information that are displayed after the tbody. The `tfoot` tag can have zero, one, or more colspec tags and one or more row.

tbody

The `tbody` tag identifies the body of a tgroup or entrytbl. The `tbody` tag must have one or more row tags.

row

The `row` tag identifies the row of information in a thead, tbody, or tfoot. The `row` tag must have one or more entry or entrytbl.

entrytbl

The `entrytbl` tag takes the place of an entry, but fits into a single row of tbody in a tgroup. The content model is the same as that of a tgroup except that tfoot is ommitted and `entrytbl` is self-excluding. Its children tags are zero, one, or more colspec or spanspec, zero or one thead or tfoot, and one tbody tag.

entry

The `entry` tag identifies an entry in a row. The `entry` tag contains ASCII text and zero, one, or many cite, term, cnxn, link, code, emphasis, or media.

Using CALS Tables

It might sound a little confusing but I think that the best way to understand a table is to look at [link]. For more information, consult the CALS Table Spec or the CNXML Spec.

Example:

```
<table id='grilltemp' frame='all'>
  <title>Steak Cooking Temperatures</title>
  <tgroup cols='2' colsep='1' rowsep='1'>
    <thead>
      <row>
        <entry>Temperature (&deg;F)</entry>
        <entry>Description</entry>
      </row>
    </thead>
    <tbody>
      <row>
        <entry align='center'>140</entry>
        <entry align='center'>Rare</entry>
      </row>
      <row>
        <entry align='center'>150</entry>
        <entry align='center'>Medium Rare</entry>
      </row>
      <row>
        <entry align='center'>160</entry>
        <entry align='center'>Medium</entry>
      </row>
      <row>
        <entry align='center'>165</entry>
        <entry align='center'>Medium Well</entry>
      </row>
      <row>
        <entry align='center'>170</entry>
        <entry align='center'>Well</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

Temperature (°F)	**Description**

Temperature (°F)	**Description**
140	Rare
150	Medium Rare
160	Medium
165	Medium Well
170	Well

Steak Cooking Temperatures

Conclusions

This concludes the CNXML tutorial.

CNXML Reference Extensions

A module describing the use of bibtexml with cnxml, as well as as introduction to the glossary tag.

Introduction

As an author/editor, you will often times need a way to include additional information in a document that does not actually appear in the flow of text. This information may include a glossary, and bibliographic references. There are many ways to include this type of information, but for our purposes, we have chosen to create a new CNXML tag named `glossary`, and have chosen to use an xml language called bibteXML for references. The two are described below. I have also included the glossary and bibteXML file examples in the source of this document. Scroll to the bottom of the page to see how these examples would be rendered.

BibteXML

" "BibteXML is a bibliography DTD for XML that expresses the content model of BibTeX, the bibliographic system for use with LaTeX. It provides conversion tools for tagging your bibliographic data in XML, or export it to HTML or native BibTeX syntax, saving typing time." " In plain language, this means that bibtexml is an XML version of the popular and widely accepted latex extension bibtex. One can markup references in their document using semantic tags such as `author` and `editor`. More info will be provided below.

The 'Glossary' Tag

Often in textbooks there will be a list of definitions included at the end of the book. In the same way, the `glossary` tag will contain a list of definitions that will be included at the end of a module. One can link to these definitions using the `term` tag (see [link]).

Including a Glossary

It is very easy to include a glossary in your CNXML document. In the Basic CNXML Tutorial it is stated that the structure usually resembles the following:
Document

- `name`
- `metadata`(optional)
- `content`

When one wishes to add a glossary the structure will change to match the following:
Document

- `name`
- `metadata`(optional)
- `content`
- `glossary`

Inside of the glossary tag one can add as many definitions as one wishes. For more information on the definition tag, see the CNXML 0.5 specification.

Example:
Glossary Example
Following is an example of the code necessary to add a glossary with one definition. `<glossary> <definition id='quardef'> <term>quarter</term> <meaning><name>Meaning Name</name>One fourth of something.</meaning> <example id='def'> <para id='par'> "He cut the pie into quarters and gave all four people a piece." </para> </example> <meaning>25 cents, a quarter of a dollar.</meaning> <example id='def2'> <para id='par2'> "The drink cost a quarter." </para>`

```
</example> <example id='def3'> <para id='par3'>
"She picked up a roll of quarters so that she
could do laundry." </para> </example>
</definition> </glossary>
```

Example:
Linking to Definitions in a Glossary
Often, one will need to refer to a definition in the glossary. To do this, one can use the `term` tag. By putting the `target-id` attribute in the term tag, one can link to a definition. Simply set the value of the `target-id` attribute to the `id` of the definition in the glossary, and that term will automatically become a link to the definition in the glossary. Shown below is an example of the term tag being used to link to the definition in the definition example: `<term target-id='quardef'>quarter</term>`

Including BibteXML

It is very easy to include a bibteXML reference section in your CNXML document. In the Basic CNXML Tutorial it is stated that the structure usually resembles the following:
Document

- `name`
- `metadata`(optional)
- `content`

When one wishes to add a bibteXML reference section the structure will change to match the following:
Document

- `name`
- `metadata`(optional)

- `content`
- `glossary`
- `file`

Note: It is possible to include either a glossary or a bibteXML file or both. The only restriction is that if you include both the glossary must precede the bibteXML file.

The `file` tag is the root tag of the bibteXML language. Inside of the file tag one will add other tags that correspond to the different type of bibliographical references. An example of a bibliography is given below:

Example:
BibteXML Example

```
<bib:file> <bib:entry id="esbensen"> <bib:book>
<bib:author>Kim Esbensen; Tonje Midtgaard; Suzanne
Schonkopf</bib:author> <bib:title>Multivariate
Analysis in Practice</bib:title>
<bib:publisher>Camo AS</bib:publisher>
<bib:year>1994</bib:year>
<bib:address>Trondheim</bib:address> </bib:book>
</bib:entry> <bib:entry id="martens.nes">
<bib:book> <bib:author>Harald Martens; Tormod
Nas</bib:author> <bib:title>Multivariate
Calibration</bib:title> <bib:publisher>John Wiley
&amp; Sons Ltd.</bib:publisher>
<bib:year>1989</bib:year>
<bib:address>Chichester</bib:address> </bib:book>
</bib:entry> </bib:file>
```

Note: Each tag in the example code begins with a namespace prefix. In the case of bibteXML, the prefix is `bib:`.

Example:
Linking to Bibliography
You will want to refer to an entry in the bibliography. To do this, one can use the `cite` tag. By putting the `target-id` attribute in the cite tag, one can link to a bibliographic entry. Simply set the value of the `target-id` attribute to the `id` of the `bib:entry`, and that reference will automatically become a link to the bibliographic entry. Shown below is an example of the cite tag being used to link to the bibliography in [link]:

```
<cite target-id='esbensen'>Multivariate Analysis
in Practice</cite>
```

BibteXML Tags

By looking at [link] one can see the types of tags that are available for use in bibteXML. Below I will attempt to give a brief explanation on the use of some of the bibteXML tags. For more information and the complete specification please see BibteXML Homepage.

File

The `file` tag is the root tag of the bibteXML language. It denotes the beginning of the bibliography.

Children
The `file` tag must contain one or more `entry` tags.

Entry

The `entry` tag denotes the beginning of an individual bibliographical reference.

Children
The `entry` must contain one of the following containers:

- article
- book
- booklet
- manual
- techreport
- mastersthesis
- phdthesis
- inbook
- incollection
- proceedings
- inproceedings
- conference
- unpublished
- misc

Containers

Each of the possible children of `entry` (article, book, booklet, etc.) are containers for metadata on that specific type of entry.

Children
Every container tag must contain a different combination of bibteXML `metadata tags`. For more information on exactly which children a specific container may possess, please see the BibteXMLHomepage.

BibteXML Metadata Tags

Every child listed as a possible child of `entry` can contain metadata tags. These metadata tags are now listed.

Metadata Tag List

- address
- author
- booktitle
- chapter
- edition
- editor
- howpublished
- institution
- journal
- month
- note
- number
- organization
- pages
- publisher
- school
- series
- title
- type
- volume
- year

Children

Every metadata tag can contain unicode text.

Glossary

quarter
Meaning Name
One fourth of something.
    25 cents, a quarter of a dollar.

Example:
"He cut the pie into quarters and gave all four people a piece."

Example:
"The drink cost a quarter."

Example:
"She picked up a roll of quarters so that she could do laundry."

Content MathML

The authoritative reference for Content MathML is Section 4 of the MathML 2.0 Specification. The World Wide Web Consortium (W3C) is the body that wrote the specification for MathML. The text is very readable and it is easy to find what you are looking for. Look there for answers to questions that are not answered in this tutorial or when you need more elaboration. This tutorial is based on MathML 2.0.

In this document, the `m` prefix is used to denote tags in the MathML namespace. Thus the `<apply>` tag is referred to as `<m:apply>`. Remember all markup in the MathML namespace must be surrounded by `<m:math>` tags.

The Fundamentals of Content MathML: Applying Functions and Operators

The fundamental concept to grasp about Content MathML is that it consists of applying a series of functions and operators to other elements. To do this, Content MathML uses prefix notation. **Prefix notation** is when the operator comes first and is followed by the operands. Here is how to write "2 plus 3". `<m:math> <m:apply> <m:plus/> <m:cn>2</m:cn> <m:cn>3</m:cn> </m:apply> </m:math>` This would display as $2+3$.

There are three types of elements in the Content MathML example shown above. First, there is the `apply` tag, which indicates that an operator (or function) is about to be applied to the operands. Second, there is the function or operator to be applied. In this case the operator, `plus`, is being applied. Third, the operands follow the operator. In this case the operands are the numbers being added. In summary, the apply tag applies the function (which could be sin or f, etc.) or operator (which could be plus or minus, etc.) to the elements that follow it.

Tokens

Content MathML has three tokens: `ci`, `cn`, and `csymbol`. A **token** is basically the lowest level element. The tokens denote what kind of element you are acting on. The **cn** tag indicates that the content of the tag is a number. The **ci** tag indicates that the content of the tag is an identifier. An **identifier** could be any variable or function; x, y, and f are examples of identifiers. In addition, `ci` elements can contain Presentation MathML. Tokens, especially `ci` and `cn`, are used profusely in Content MathML. Every number, variable, or function is marked by a token.

csymbol is a different type of token from `ci` and `cn`. It is used to create a new object whose semantics is defined externally. It can contain plain text or Presentation MathML. If you find that you need something, such as an operator or function, that is not defined in Content MathML, then you can use csymbol to create it.

Both `ci` and `csymbol` can use Presentation MathML to determine how an identifier or a new symbol will be rendered. To learn more about Presentation MathML see Section 3 of the MathML 2.0 Specification. For example, to denote "x with a subscript 2", where the 2 does not have a more semantic meaning, you would use the following code. `<m:math> <m:ci> <m:msub> <m:mi>x</m:mi> <m:mn>2</m:mn> </m:msub> </m:ci> </m:math>` This would display as x_2.

The `ci` elements have a type attribute which can be used to provide more information about the content of the element. For example, you can declare the contents of a `ci` tag to be a function (`type='fn'`), or a vector (`type='vector'`), or a complex number (`type='complex'`), as well as any number of other things. Using the type attribute helps encode the meaning of the math that you are writing.

Functions and Operators

In order to apply a function to a variable, make the function the first argument of an apply. The second argument will be the variable. For example, you would use the following code to encode the meaning, "the function f of x". (Note that you have to include the attribute `type='fn'` on the `ci` tag denoting f.) `<m:math> <m:apply> <m:ci type='fn'>f</m:ci> <m:ci>x</m:ci> </m:apply> </m:math>` This will display as $f(x)$.

There are also pre-defined functions and operators in Content MathML. For example, sine and cosine are predefined. These predefined functions and operators are all **empty tags** and they directly follow the apply tag. "The sine of x" is similar to the example above. `<m:math> <m:apply> <m:sin/> <m:ci>x</m:ci> </m:apply> </m:math>` This will display as $\sin(x)$.

You can find a more thorough description of the different predefined functions in Chapter 4 of the MathML specification.

In addition to the predefined functions, there are also many predefined operators. A few of these are `plus` (for addition), `minus` (for subtraction), `times` (for multiplication), `divide` (for division), `power` (for taking the nth-power of something), and root (for taking the nth-root of something).

Most operators expect a specific number of child tags. For example, the power operator expects two children. The first child is the base and the second is the value in the exponent. However, there are other tags which can take many children. For example, the plus operator merely expects one or more children. It will add together all of its children whether there are two or five. This is referred to as an **n-ary operator**.

Representing "the negative of a variable" and explicitly representing "the positive of a variable or number" has slightly unusual syntax. In this case you apply the plus or minus operator to the variable or number, etc., in question. The following is the code for "negative x." `<m:math> <m:apply> <m:minus/> <m:ci>x</m:ci> </m:apply> </m:math>` This will display as $-x$.

In contrast to representing the negative of a variable, the negative of a number may be coded as follows: `<m:math><m:cn>-1</m:cn> </m:math>` This will display as -1.

To create more complicated expressions, you can nest these bits of apply code within each other. You can create arbitrarily complex expressions this way. "a times the quantity b plus c" would be written as follows. `<m:math> <m:apply> <m:times/> <m:ci>a</m:ci> <m:apply> <m:plus/> <m:ci>b</m:ci> <m:ci>c</m:ci> </m:apply> </m:apply> </m:math>` This will display as $a\left(b+c\right)$.

The `eq` operator is used to write equations. It is used in the same way as any other operator. That is, it is the first child of an apply. It takes two (or more) children which are the two quantities that are equal to each other. For example, "a times b plus a times c equals a times the quantity b plus c" would be written as shown. `<m:math> <m:apply> <m:eq/> <m:apply> <m:plus/> <m:apply> <m:times/> <m:ci>a</m:ci> <m:ci>b</m:ci> </m:apply> <m:apply> <m:times/> <m:ci>a</m:ci> <m:ci>c</m:ci> </m:apply> </m:apply> <m:apply> <m:times/> <m:ci>a</m:ci> <m:apply> <m:plus/> <m:ci>b</m:ci> <m:ci>c</m:ci> </m:apply> </m:apply> </m:apply> </m:math>` This will display as $ab+ac=a\left(b+c\right)$.

Integrals

The operator for an integral is `int`. However, unlike the operators and functions discussed above, it has children that define the independent variable that you integrate with respect to (`bvar`) and the interval over which the integral is taken (use either `lowlimit` and `uplimit`, or `interval`, or `condition`). `lowlimit` and `uplimit` (which go together), `interval`, and `condition` are just three different ways of denoting the integrands. Don't forget that the bvar, `lowlimit`, `uplimit`, `interval`, and `condition` children take token elements as well. The following is "the integral of f of x with respect to x from 0 to b." `<m:math> <m:apply> <m:int/> <m:bvar><m:ci>x</m:ci> </m:bvar> <m:lowlimit><m:cn>0</m:cn></m:lowlimit> <m:uplimit><m:ci>b</m:ci></m:uplimit> <m:apply>`

`<m:ci type='fn'>f</m:ci> <m:ci>x</m:ci> </m:apply> </m:apply> </m:math>` This will display as $\int_0^b f(x)\,\mathrm{d}\,x$.

Derivatives

The derivative operator is `diff`. The derivative is done in much the same way as the integral. That is, you need to define a base variable (using `bvar`). The following is "the derivative of the function f of x, with respect to x." `<m:math> <m:apply> <m:diff/> <m:bvar> <m:ci>x</m:ci> </m:bvar> <m:apply> <m:ci type="fn">f</m:ci> <m:ci>x</m:ci> </m:apply> </m:apply> </m:math>` This will display as $\frac{\mathrm{d}}{\mathrm{d}x} f(x)$.

To apply a higher level derivative to a function, add a `degree` tag inside of the `bvar` tag. The degree tag will contain the order of the derivative. The following shows "the second derivative of the function f of x, with respect to x." `<m:math> <m:apply> <m:diff/> <m:bvar> <m:ci>x</m:ci> <m:degree><m:cn>2</m:cn></m:degree> </m:bvar> <m:apply><m:ci type="fn">f</m:ci> <m:ci>x</m:ci> </m:apply> </m:apply> </m:math>` This will display as $\frac{\mathrm{d}^2}{\mathrm{d}x^2} f(x)$.

Vector and Matrices

Vectors are created as a combination of other elements using the `vector` tag. `<m:math> <m:vector> <m:apply> <m:plus/> <m:ci>x</m:ci> <m:ci>y</m:ci> </m:apply> <m:ci>z</m:ci> <m:cn>0</m:cn> </m:vector> </m:math>` This will display as $\begin{matrix} x+y \\ z \\ 0 \end{matrix}$.

Matrices are done in a similar manner. Each `matrix` element contains several `matrixrow` elements. Then each `matrixrow` element contains several other elements. `<m:math> <m:matrix> <m:matrixrow> <m:ci>a</m:ci> <m:ci>b</m:ci> <m:ci>c</m:ci> </m:matrixrow> <m:matrixrow> <m:ci>d</m:ci> <m:ci>e</m:ci> <m:ci>f</m:ci> </m:matrixrow> <m:matrixrow> <m:ci>g</m:ci> <m:ci>h</m:ci> <m:ci>j</m:ci> </m:matrixrow> </m:matrix> </m:math>` This will display as $\begin{matrix} a & b & c \\ d & e & f \\ g & h & j \end{matrix}$.

There are also operators to take the determinant and the transpose of a matrix as well as to select elements from within the matrix.

Entities

Note: The use of MathML character entity references in Connexions content is **deprecated**.

MathML defines its own entities for many special characters used in mathematical notation. While the entity references have the advantage of being mnemonic with respect to the characters they stand for, they also entail some technical limitations, and so their use in Connexions content is deprecated. Please use the UTF-8-encoded Unicode characters themselves where possible, or, failing that, the XML Unicode character references for the characters. At some time in the future, the Connexions repository system will likely convert entity references and character references silently to the UTF-8-encoded Unicode characters they stand for. See 6.2.1 Unicode Character Data from the XML Specification for more information. The MathML specification contains a list of character entities with their corresponding Unicode code points.

There are character picker utilities available to help you select and paste UTF-8 characters into applications like Connexions. If you are running

Microsoft Windows, the Windows accessory Character Map can help you. The "Lucida Sans Unicode" font seems to have a good selection of mathematical operators and special characters. Under Linux, the charmap utility and GNOME applet provide access to all Unicode characters.

Other Resources

There is a lot more that can be done with Content MathML. Especially if you are planning on writing a lot of Content MathML, it is well worth your time to take a look at the MathML specification.

QML 1.0 tutorial
Tutorial for QML 1.0

Note: This tutorial currently under revision and may contain errors or inconsistencies in its current state. If you have any questions or notice any problems with this module, please contact techsupport@cnx.org and let us know.

This tutorial will show you how to create problemsets and write individual items, and how to embed them into other XML documents. It is intended primarily for people involved in the Connexions Project.

Displaying QML

The Connexions project provides the ability to do simple response processing through stylesheets and JavaScript. Examples of QML items thus processed are available in the QML 1.0 examples module.

PLEASE NOTE: This response processing is intended for the student to do self-testing as he or she moves through modules. Any student with minor technological know-how can determine the answers to the questions by viewing the source file. Hence, if you do not want students to have easy access to the answers to the questions, do not include them in the QML - leave the key blank and do not put information in the feedback tags that will allow the student to determine the correct answer.

Please also note that while the current Connexions response processing can determine whether answers to single-response, multiple-response, and single-response questions are correct, it does not process the responses to text-response questions (this requires high-level and very situation-specific software that we do not have plans to provide at this time). Instead, it only shows the general feedback.

Note: In previous versions of the CNXML language it was necessary to modify the document tags to include the use of QML. As of CNXML 0.6 this step is no longer required, as QML is supported natively within CNXML documents.

Structural overview

In QML, items (test and homework questions) are either grouped together in a problemset or are written individually. Each item contains the question being asked of the user, the response options together with answer-specific feedback, if any, general feedback, if any, and a key. The items can also contain hints and links to resources.

Problemsets and items

The first thing you need to do is decide whether your items should be grouped together in a problemset or not. If you are writing QML with CNXML, you can only use items, and each item will go within a CNXML exercise tag. If you have a reason to logically group your items together (e.g. they comprise a homework set) and you are not writing for CNXML, enclose them in a problemset. To write a problemset, all you need to do is enclose your individual items within a problemset tag. If you are writing individual items and do not have a good reason to group them together within problemsets, or you are using CNXML, write items without enclosing them within a problemset tag.

Problemset

The problemset tag encloses one or more items, and has one optional attribute called id. If you will be making a number of problemsets and need to be able to reference them individually, you will want to add a unique id attribute to each of your problemsets to distinguish between them.

Example

```
<problemset id="homework1">

  <item . . .>
  </item>

</problemset>
```

Item

The next tag is the item tag, which contains the question, response options, and so forth. Each item tag must contain an id attribute and a type attribute. Ideally, the id attribute should be unique; no two items you write should have the same attribute. How you do this is up to you.

The type attribute determines the type of question you are asking. It must be one of the following:

- single-response
- multiple-response
- ordered-response
- text-response

Finally, it is important to note that an item must contain one question tag. It may also contain zero or more answer tags, and zero or one key tags. However, the existence or non-existence, as well as the contents of these last two tags, may be slightly different depending on the type of question. Items may also contain one or more resource tags and hint tags, and one general feedback tag.

Single-response

Single-response items are items for which there is one and only one correct answer to be selected from the provided answers. Examples are multiple-choice (select only one) and Likert scale items.

Multiple-response

Multiple-response items are items which require more than one selected response option for a correct answer. Examples are multiple-choice items where the user must select two or more responses to each item.

Ordered-response

Ordered-response items are multiple-response items for which the order of the selected response is important. An example is an item that asks the user to select the instructions for performing a procedure in their proper order.

Text-response

Text-response items are items which require a textual or numeric response. Examples are short-answer questions and supply-the-answer math questions.

Example

```
  An example for a single-response question:

<item id="item1" type="single-response">

  <question>
```

```
          Are bananas a fruit or a vegetable?
        </question>

        <resource
uri="http://bananas.com/bananas.csv" />

        <answer id="fruit">
          .
          .
        </answer>

        <answer id="vegetable">
          .
          .
        </answer>

        <hint>Bananas grow on trees.</hint>

        <feedback>Bananas are a fruit.</feedback>

        <key answer="fruit" />

      </item>
```

Question

The question tag contains the question you are asking of the user. It can be written in plain text, or using markup from another language (such as HTML, which allows you to include pictures and graphs as well as text). Be aware that the stylesheets for the Connexions project currently only allow plain text and CNXML (with and without MathML) within question tags. If you use CNXML (with or without MathML) within a QML tag, the first tag must be a section tag, which may contain any of the tags a CNXML section tag is allowed to contain.

Example

```
      <question>
        Are bananas a fruit or a vegetable?
      </question>
```

Resource

The resource tag allows you to provide a URI attribute (file location, web address, etc.) for a particular item. For example, if you had the necessary software, you could use the resource tag to attach a reference to a dataset and generate question and answer values from that dataset. At the present time, Connexions has no software to do this, and the resource tag is ignored by our stylesheets.

It also has an optional id attribute. If you are using more than one resource for a particular item and need to keep track of which resource is which, you will want to include an id tag.

The resource tag is an empty tag, which means you must include a slash at the end of the tag.

Example

```
      <resource
uri="http://cnx.rice.edu/datasets/dataset1.csv"
id="resource1" />
```

Answer, Response, and Feedback

How you construct the answer tag, including whether you include an answer tag at all, depends on your item-type. However, in all cases where you have an answer tag, the ID of the answer tag must not contain any commas, as the key tag refers to the answer IDs separated by commas.

Single-response, multiple-response, and ordered-response

To write answer tags for single-response, multiple-response, and ordered-response items:

You will want one answer tag for each response option. Each answer tag has an id, which is referenced by the key tag, below. Preferably, the id tags should have an obvious relation to the response option. For example, if your response options are "London, England;" "Washington, DC;" and "Paris, France" your id attributes could be london, washington, and paris.

Each of your answer tags should contain a response tag and may contain a feedback tag if you wish to provide feedback specific to that response (general feedback goes in another tag). The response tag contains the response option, in text or markup, and has no attributes. Be aware that the stylesheets for the Connexions project currently only allow plain text and CNXML (with or without MathML) within response tags. If you use CNXML (with or without MathML) within a QML tag, the first tag must be a section tag, which may contain any of the tags a CNXML section tag is allowed to contain.

The feedback tag contains feedback to be displayed to the user upon selection of that response option. The feedback may be text or markup. The feedback tag does not have attributes for single-response, multiple-response, or ordered-response items.

Example for a single-response question

```
        <item id="item1" type="single-response">

          <question>
            Are bananas a fruit or a vegetable?
          </question>

          <resource
uri="http://bananas.com/banana_dataset1.csv" />

          <answer id="fruit">
            <response>A fruit.</response>
            <feedback>Correct!</feedback>
          </answer>

          <answer id="vegetable">
            <response>A vegetable.</response>
            <feedback>Incorrect.</feedback>
          </answer>

          <hint>Bananas grow on trees.</hint>

          <feedback>Bananas are a fruit.
</feedback>

          <key answer="fruit" />

      </item>
```

Text-response items

To write answer tags for text-response items: Because text-response items do not have response options, you will have either have no answer tag (if you are not providing response feedback) or only one answer tag which may contain two response-specific feedback tags: one for a correct response, one for an incorrect response.

The feedback tag containing the correct feedback should have the attribute correct="yes", and the other should have the attribute correct="no". You may also include a general feedback tag; this is recommended as Connexions does not have plans to provide response processing for text-response items and at the present time only shows the general feedback for the user to compare his/her answer with. The content of the feedback tags may be text or markup; be aware that the stylesheets for the Connexions project currently only allow plain text and CNXML (with or without MathML) within question tags. If you use CNXML (with or without MathML) within a QML tag, the first tag must be a section tag, which may contain any of the tags a CNXML section tag is allowed to contain.

Example for a text-response item

```
        <item id="item1" type="text-response">

          <question>
            Are bananas a fruit or a vegetable?
          </question>

          <resource
uri="http://bananas.com/bananas2.csv" />

          <answer id="fruit">
            <feedback correct="yes">Correct!
</feedback>
            <feedback correct="no">Incorrect.
</feedback>
          </answer>
          <hint>Bananas grow on trees.</hint>

          <feedback>Bananas are a fruit.
</feedback>

          <key answer="fruit" />

        </item>
```

Hint

The hint tags each contain a hint to be displayed to the user upon request. The content of the hint tags is text or markup. Be aware that the stylesheets for the Connexions project currently only allow plain text and CNXML (with or without MathML) within response tags. If you use CNXML (with or without MathML) within a QML tag, the first tag must be a section tag, which may contain any of the tags a CNXML section tag is allowed to contain. The hint tag has no attributes; it is assumed that if there are multiple hints, they will be displayed to the user in order.

Examples

```
        <hint>Bananas grow on trees.</hint>
```

Key

The key tag contains the answer key, for items for which the answer key is included.

The key tag varies slightly depending on the type of item. For single-response items, the key tag contains an "answer" attribute which refers to the id tag of the correct answer.

Example

```
<key answer="fruit" />
```

For multiple-response items, the answer id should refer to all the necessary response ids for a correct answer, separated by commas.

Example

```
<key answer="fruit,yellow" />
```

For ordered-response items, the response ids should be in correct order and separated by commas.

Example

```
<key answer="uno,dos,tres,cuatro,cinco" />
```

For text-response items, the key tag should have no attributes, and should contain text which will be helpful to the program or person scoring the user's answers.

Examples

```
      <key>One example of a correct answer is "No
hay nadie
      aqui."</key>

      <key>3.14159</key>
```

Complete examples

Stripped-down single-response item

Here is an example of a single-response item with only a question, two answers, and a key.

```
        <item id="item1" type="single-response">

          <question>
            Are bananas a fruit or a vegetable?
          </question>

          <answer id="fruit">
            <response>A fruit.</response>
          </answer>
          <answer id="vegetable">
            <response>A vegetable.</response>
          </answer>

          <key answer="fruit" />

        </item>
```

Full single-response item

Here is an example of a single-response item with all options - resource, hints, and feedback.

```
        <item id="item1" type="single-response">

          <question>
            Are bananas a fruit or a vegetable?
          </question>

          <resource
uri="http://bananas.com/yellowbananas.csv" />

          <answer id="fruit">
            <response>A fruit.</response>
            <feedback>Correct!</feedback>
          </answer>

          <answer id="vegetable">
            <response>A vegetable.</response>
            <feedback>Incorrect.</feedback>
          </answer>
          <hint>Bananas grow on trees.</hint>

          <feedback>Bananas are a fruit.
</feedback>

          <key answer="fruit" />

        </item>
```

Stripped-down multiple-response item

Here is an example of a multiple-response item with only a question, three answers, and a key.

```
        <item id="item1" type="multiple-response">

          <question>
            Bananas are (pick two):
          </question>

          <answer id="fruit">
            <response>A fruit.</response>
          </answer>

          <answer id="vegetable">
            <response>A vegetable.</response>
          </answer>

          <answer id="yellow">
            <response>Yellow.</response>
          </answer>
          <key answer="fruit,yellow" />

        </item>
```

Full multiple-response item

Here is an example of the same multiple-response item with all options - resource, hints, and feedback.

```
        <item id="item1" type="multiple-response">

          <question>
            Bananas are (pick two):
          </question>

          <resource
uri="http://bananas.com/bananas.csv" />

          <answer id="fruit">
            <response>A fruit.</response>
            <feedback>Yes, bananas are a fruit.
</feedback>
          </answer>

          <answer id="vegetable">
            <response>A vegetable.</response>
            <feedback>Bananas are not a vegetable.
</feedback>
          </answer>

          <answer id="yellow">
            <response>Yellow.</response>
            <feedback>Yes, bananas are yellow.
</feedback>
          </answer>

          <hint>Bananas grow on trees.</hint>
          <hint>Bananas are the same color as
lemons.</hint>

          <feedback>Bananas are a yellow fruit.
</feedback>

          <key answer="fruit,yellow" />

        </item>
```

Stripped-down ordered-response item

Here is an example of an ordered-response item with only a question, three answers, and a key.

```
        <item id="item1" type="ordered-response">

          <question>
            In order to boil water, you need to do
the following
            (please select in the correct order):
          </question>

          <answer id="stove">
            <response>Put the pot on the stove.
</response>
          </answer>
          <answer id="pot">
            <response>Put water in the pot.
</response>
          </answer>

          <answer id="boil">
            <response>Wait until you see bubbles
in the water.</response>
          </answer>

          <key answer="pot,stove,boil" />

        </item>
```

Full ordered-response item

Here is an example of the same ordered-response item with all options - resource, hints, and feedback.

```
        <item id="item1" type="ordered-response">

          <question>
            In order to boil water, you need to do
the following
            (please select in the correct order):
          </question>

          <resource
uri="www.howtoboilwater.com/instructions.csv" />

          <answer id="stove">
```

```
            <response>Put the pot on the stove.
</response>
            <feedback>You put the pot on the stove
second.</feedback>
          </answer>

          <answer id="pot">
            <response>Put water in the pot.
</response>
            <feedback>You put the water in the pot
first.</feedback>
          </answer>

          <answer id="boil">
            <response>Wait until you see bubbles
in the water.</response>
            <feedback>You wait for the bubbles
third.</feedback>
          </answer>

          <hint>The pot won't boil till there's
water in it.</hint>

          <feedback>First you put water in the
pot, then you put the
          pot on the stove, then you wait for the
bubbles.</feedback>

          <key answer="pot,stove,boil" />

        </item>
```

Stripped-down text-response item

Here is an example of an ordered-response item with only a question, general feedback, and a key.

```
        <item id="item1" type="text-response">

          <question>
            What are some common tests of
executive function?
          </question>

          <feedback>Some common tests of executive
function are the
          Wisconsin Card Sort Test and the Tower
of Hanoi.</feedback>

          <key>Wisconsin Card Sort Test, Tower of
Hanoi</key>

        </item>
```

Full text-response item

Here is an example of an ordered-response item with a resource, hints, and an answer with correct and incorrect feedback.

```
        <item id="item1" type="text-response">

          <question>
            What are some common tests of
executive function?
```

```
          </question>

          <resource
uri="www.execfunc.com/tests.csv" />

          <answer>
            <feedback correct="no">Incorrect.
</feedback>
            <feedback correct="yes">Correct.
</feedback>
          </answer>

          <hint>Think about square states and
towers.</hint>

          <feedback>Some common tests of executive
function are the
          Wisconsin Card Sort Test and the Tower
of Hanoi.</feedback>

          <key>Wisconsin Card Sort Test, Tower of
Hanoi</key>

        </item>
```

Grilling a Good Steak

I have eaten many steaks in my life and none have been more satisfying than the backyard-grill cooked steak. Maybe this is because of the relaxing nature of drinking a beer, being outside, and lounging that accompanies the grilling procedure.

Note: Excessive drinking or fun may result in overcooked or burned steak

Maybe it is because of the aroma of the grill and the beef **perfectly** seasoned to your taste. Either way, this module shows how a good steak can be prepared.

Before we begin to cook I have compiled a list of ingredients.
Ingredients

- Salt
- Fresh ground pepper
- Lime
- Beer
- Chili powder
- T-Bone steak

To ensure the best flavor possible, it is necessary to marinate the beef first. A steak **marinates** when left to sit in **marinade**, or prepared sauce, where it will absorb the flavor of the ingredients. Marinating may take as little as 15 minutes or as long as 6 hours and should **always** be done in the refrigerator and **not** at room temperature.
Instructions

1. pour beer into large bowl
2. add chili powder to tase
3. squeeze half lime into beer marinade
4. place steak in beer, let soak for 30 minutes

I'll be adding to this module in The Intermediate CNXML which focuses on more advanced CNXML tags. For more marinades see the Angus Beef website. Finally, a good resource is the *Steak Lover's Cookbook* -- *William Rice*.

Grilling a Better Steak

I have eaten many steaks in my life and none have been more satisfying than the backyard-grill cooked steak. Maybe this is because of the relaxing nature of drinking a beer, being outside, and lounging that accompanies the grilling procedure. Maybe it is because of the aroma of the grill and the beef perfectly seasoned to your taste. Either way, this module shows how a good steak can be prepared.

Ingredients

Before we begin to cook I have compiled a list of ingredients.
Ingredients

- Salt
- Fresh ground pepper
- Lime
- Beer
- Chili powder
- T-Bone steak

Marinade

To ensure the best flavor possible, it is necessary to marinate the beef. A steak **marinates** when left to sit in **marinade**, or prepared sauce, where it will absorb the flavor of the ingredients. Marinating may take as little as 15 minutes or as long as 6 hours and should **always** be done in the refrigerator and **not** at room temperature.
Marinade

1. pour beer into large bowl
2. add chili powder to tase
3. squeeze half lime into beer marinade
4. place steak in beer, let soak for 30 minutes

T-Bone Steak

Upon successful completion of these modules, you should be able to grill a steak that looks just as good!

Steaks

T-Bone

New York Strip

Upon successful completion of these modules, you should be able to grill a steak that looks just as good!

Steaks

Upon successful completion of these modules, you should be able to grill a steak that looks just as good!

T-Bone

New York Strip

How to grill the steak will be covered in The Advanced CNXML. For more marinades see the Angus Beef website. Finally, a good resource is the *Steak Lover's Cookbook* -- *William Rice; Paperback*.

Grilling the Best Steak

I have eaten many steaks in my life and none have been more satisfying than the backyard-grill cooked steak. Maybe this is because of the relaxing nature of drinking a beer, being outside, and lounging that accompanies the grilling procedure. Maybe it is because of the aroma of the grill and the beef perfectly seasoned to your taste. Either way, this module shows how a good steak can be prepared.

Steaks

T-Bone

New York Strip

Upon successful
completion of these

modules, you should
be able to grill a steak
that looks just as
good!

Ingredients

Before we begin to cook I have compiled a list of ingredients.
Ingredients

- Salt
- Fresh ground pepper
- Lime
- Beer
- Chili powder
- T-Bone

T-Bone
> "The T-bone steak is cut between 1 and 3 inches thick and comes from the center section of the short loin. This steak is characterized by its T-shape bone, has a fine-grained shell and a small tenderloin eye," *http://www.chophousecalgary.com/steak.html*.

Marinade

To ensure the best flavor possible, it is necessary to marinate the beef. A steak **marinates** when left to sit in a prepared sauce, or **marinade**, where it will absorb the flavors of the ingredients. Marinating may take as little as 15 minutes or as long as 6 hours and should **always** be done in the refrigerator and **not** at room temperature.
Marinade

1. pour beer into large bowl
2. add chili powder to taste
3. squeeze half lime into beer marinade

4. place steak in beer, let soak for 30 minutes
5. before grilling rub salt and pepper onto steak

Grilling

Grilling is pretty easy. After having heated the coals or igniting the grill, start cooking the meat. I would recommend periodically checking the meat and when you start to see it being cooked on top, flip it over. Then, wait until fully cooked. Below you will find a table of cooking temperatures. Please note the safety warning at the bottom.

Temperature(F)	Description
140	Rare
150	Medium Rare
160	Medium
165	Medium Well
170	Well

Remember that for safety's sake, always cook your steak to 160 F or until meat is no longer pink.

Rounding Off the Experience

The experience of grilling a steak in your own back yard is part of what makes the home cooked steak so enjoyable. It is necessary to cook in the evening as it is getting cool and to enjoy your beverage of choice. Finally, one of the best ways to enjoy a steak is in the company of your friends.

To make sure that you were paying attention to my tutorial, I've included a one question exam:

Exercise:

Problem:

For food safety, a steak should be cooked to a minimum temperature of what?

Solution:

160 F or until the juices run clear and the meat is no longer pink

For more marinades see the Angus Beef website. Finally, a good resource is the *Steak Lover's Cookbook* -- *William Rice; Paperback*.